MOTOROLA
Semiconductor Products Inc.

AN-866
Application Note

VECTORING BY DEVICE USING INTERRUPT SYNC ACKNOWLEDGE WITH THE MC6809/MC6809E

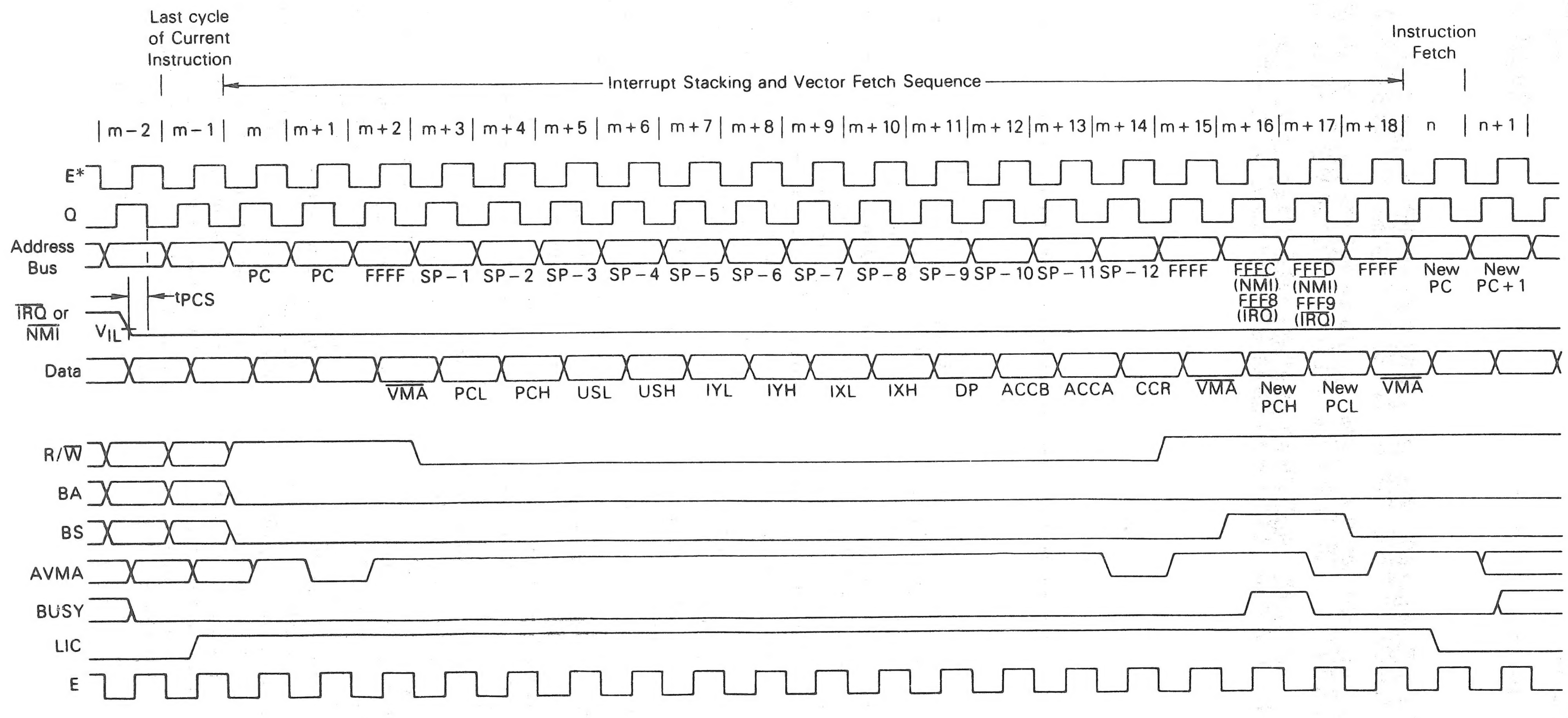
Interrupt routines have many advantages, one of which is the ability to synchronize the MPU to external events. Response time for interrupts can be very lengthy. For instance, stacking registers require 9 cycles for $\overline{\text{FIRQ}}$ and 18 for $\overline{\text{NMI}}$ or $\overline{\text{IRQ}}$. Also, the current instruction must be completed before the stacking operation can begin which further increases response time. Once the stacking operation is complete, the MPU fetches the appropriate interrupt vector and jumps to an interrupt service that may have to poll various devices to determine which peripheral is requesting services. This has been a standard mode of operation in the use of microprocessors for some time. The MC6809/MC6809E provides a method which responds to the interrupt in a slightly different way. After an interrupt is recognized and the stacking operation has begun, the BA and BS lines are both logical 0s, indicating that the processor is in a normal state. Figure 1 details the timing for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ and Figure 2 shows the interrupt timing for $\overline{\text{FIRQ}}$. In both Figure 1 and Figure 2, note that BS = 1 and BA = 0 only during the interrupt vector fetch. Also note that the BUSY line is asserted during the high-order interrupt fetch. The BUSY and BS lines remain low during all other cycles of the interrupt response sequence. Thus, BA and BS may be used to shorten the response time required for an interrupt by allowing for hardware vectoring by device. In a vector by device scheme, the interrupting device provides all or part of the vector address, thus eliminating the polling portion of the response time.

A circuit to allow vectoring by device is shown in Figure 3. During the interrupt vector fetch, BS = 1 and BA = 0, causing the output of U1 to go low. The low-order byte of an $\overline{\text{IRQ}}$ is fetched from $FFF9. Bits A0 through A3 are used to decode the hexadecimal 9 at the output of U2, a four input NAND gate. When the outputs of both U1 and U2 are low, the output of U3 is also low, enabling buffer U5. The output of U3 is also inverted and ORed with the ROM chip select for the ROM at $F000 to $FFFF. When address $FFF8 (the high-order address byte for $\overline{\text{IRQ}}$) appears, the ROM is selected, and when $FFF9 appears, the buffer device (U5) is selected. Up to eight peripheral devices can have their $\overline{\text{IRQ}}$ lines connected to the 74LS148 priority encoder. When the MPU fetches the $\overline{\text{IRQ}}$ vector, it gets the high-order byte from ROM and the low-order byte from priority encoder U6 (via buffer U5). It is the circuit designer's responsibility to properly connect the $\overline{\text{IRQ}}$ lines to the priority encoder such that the device to be given highest priority is connected to input 7 and the lowest priority has its $\overline{\text{IRQ}}$ line connected to input 0. The three outputs from the priority encoder are then connected to the MPU data bus through buffer U5. Output 3 is connected to D4 of the data bus, output 2 to D3, and output 1 to Dn of the data bus.

The software must be written to accommodate the fact that all eight devices have the same high-order byte for their interrupt vector. The lower byte will, of course, be different. The program shown in Figure 4 shows that $FFF8, the high order $\overline{\text{IRQ}}$ vector, is set to $A0. When the low order byte is fetched from buffer U5, the range of values is limited from $00 to $1C. Long branch instructions are located at addresses $A000 to $A01C. A long branch instruction is a three byte instruction with the two post bytes being the 16-bit offset. By offsetting the three outputs of buffer U5, the consecutive numbers being outputted appear to the MPU as addresses which are four memory locations apart. The priority encoder outputs could be offset on the data bus in different ways to accommodate the desires of the programmer.

Normally all $\overline{\text{IRQ}}$ lines would be connected together. An interrupt routine would poll the status registers of each device until an interrupt bit was found set. As shown in Figure 5, 10 cycles are required to poll each status register. If the device with the lowest priority in the polling loop was called, it would take the program 70 machine cycles to respond. In the case of the vector-by-device program (Figure 4), the highest priority calling device will be responded to

Reprint October 1983



*E clock shown for reference only.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage of 2.0 volts, unless otherwise noted.

FIGURE 1 — $\overline{NMI}$ and $\overline{IRQ}$ Interrupt Timing Diagram

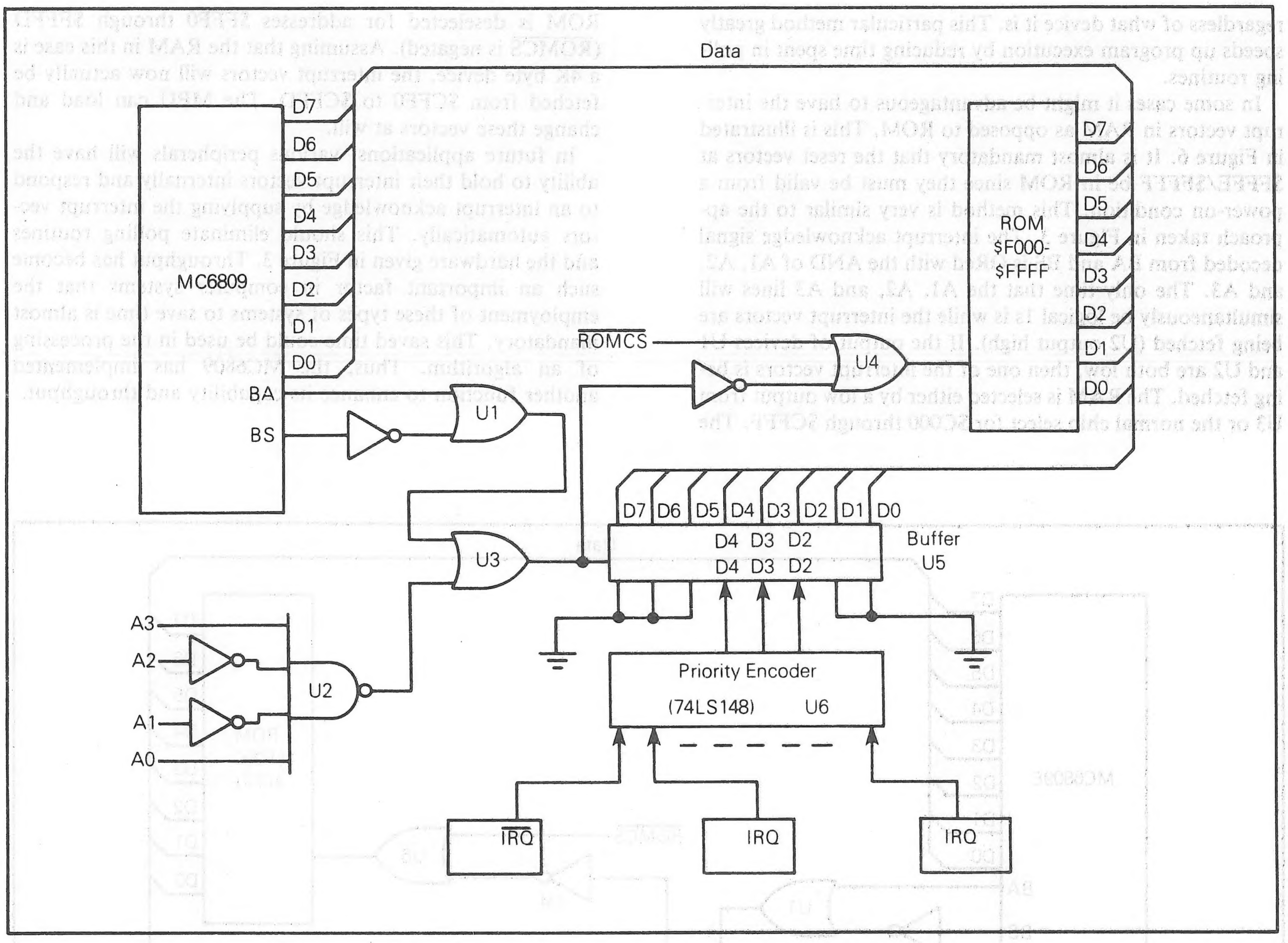


FIGURE 3 — $\overline{\text{IRQ}}$ Vectoring By Device, Block Diagram

```
        ORG     $A000
A000    LBRA    ROUTE0
A004    LBRA    ROUTE1
A008    LBRA    ROUTE2
A00C    LBRA    ROUTE3
A010    LBRA    ROUTE4
A014    LBRA    ROUTE5
A018    LBRA    ROUTE6
A01C    LBRA    ROUTE7
 •       •       •
 •       •       •
 •       •       •
 •       •       •
 •       •       •
 •       •       •
        ORG     $FFF8
        FCB     $A0
```

FIGURE 4 — Vector-By-Device Program

```
          LDB     SR#0       5 Cycles
          BITB    #______    2 Cycles
          BEQ     NEXT1      3 Cycles
          LBRA    _______
NEXT 1    LDB     SR#1
          BITB    #______
          BNE     NEXT2
          LBRA    _______
NEXT 2    LDB     SR#2
           •
           •
           •
           •
           •
          LDB     SR#7
          BITB    #______
          LBRA    _______
```

FIGURE 5 — Polling Routine For Eight Peripherals

regardless of what device it is. This particular method greatly speeds up program execution by reducing time spent in polling routines.

In some cases it might be advantageous to have the interrupt vectors in RAM as opposed to ROM. This is illustrated in Figure 6. It is almost mandatory that the reset vectors at $FFFE/$FFFF be in ROM since they must be valid from a power-on condition. This method is very similar to the approach taken in Figure 3. The interrupt acknowledge signal decoded from BA and BS is ORed with the AND of A1, A2, and A3. The only time that the A1, A2, and A3 lines will simultaneously be logical 1s is while the interrupt vectors are being fetched (U2 output high). If the output of devices U1 and U2 are both low, then one of the interrupt vectors is being fetched. The RAM is selected either by a low output from U3 or the normal chip select for $C000 through $CFFF. The ROM is deselected for addresses $FFF0 through $FFFD ($\overline{\text{ROMCS}}$ is negated). Assuming that the RAM in this case is a 4K byte device, the interrupt vectors will now actually be fetched from $CFF0 to $CFFD. The MPU can load and change these vectors at will.

In future applications, various peripherals will have the ability to hold their interrupt vectors internally and respond to an interrupt acknowledge by supplying the interrupt vectors automatically. This should eliminate polling routines and the hardware given in Figure 3. Throughput has become such an important factor in computer systems that the employment of these types of systems to save time is almost mandatory. This saved time could be used in the processing of an algorithm. Thus, the MC6809 has implemented another function to enhance its capability and throughput.

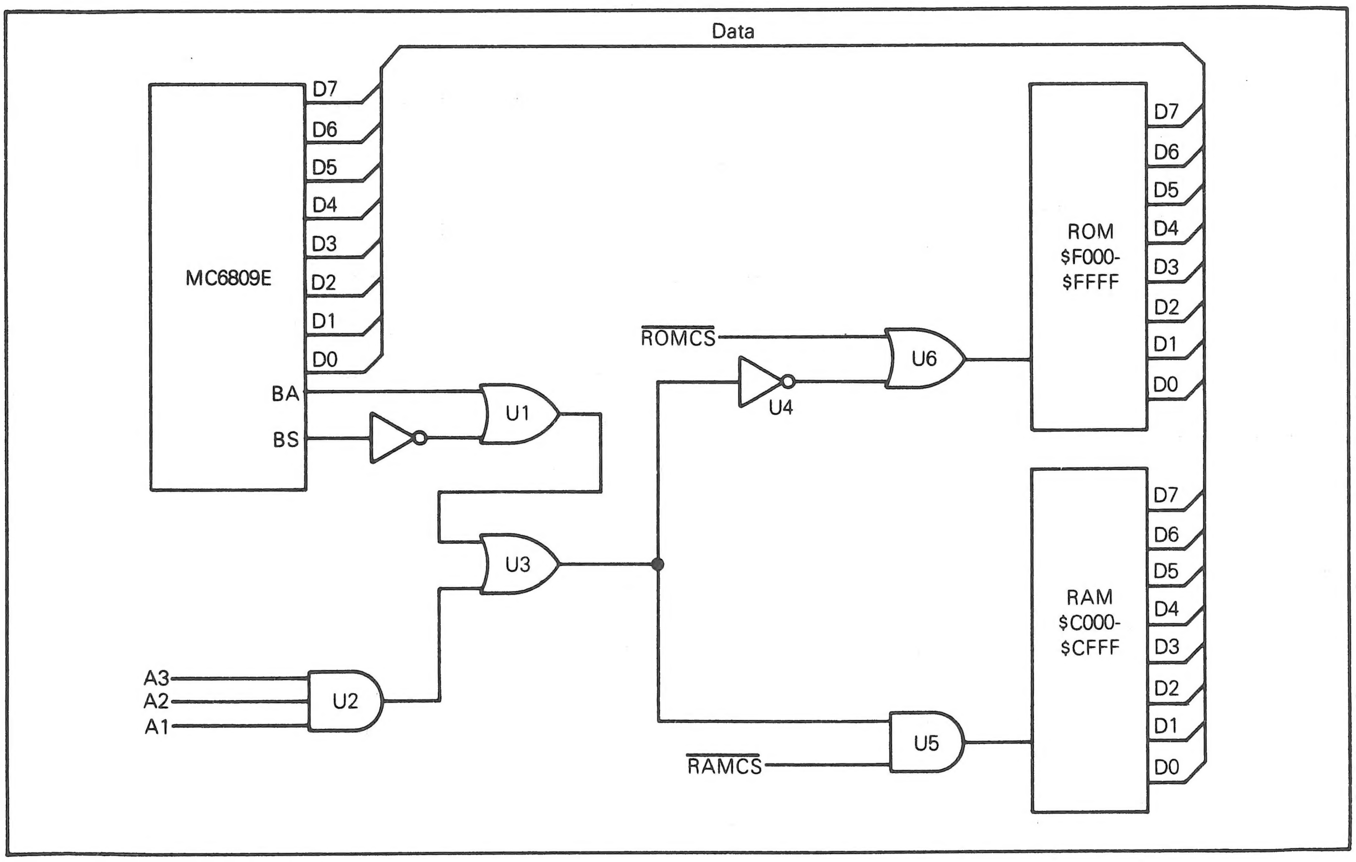
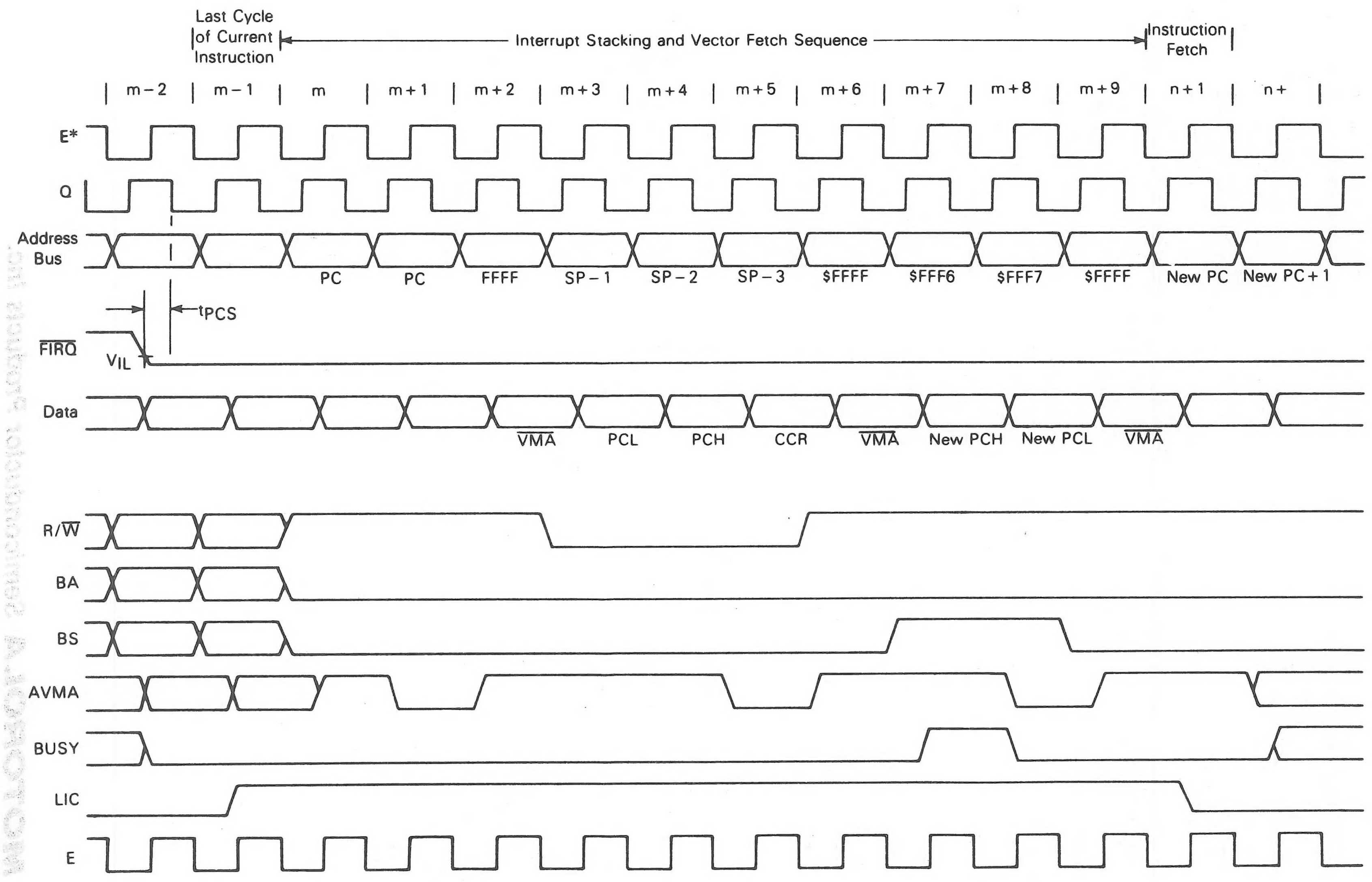


FIGURE 6 — Interrupt Vectors In RAM, Block Diagram



*E clock shown for reference only.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage of 2.0 volts, unless otherwise noted.

FIGURE 2 — $\overline{\text{FIRQ}}$ Interrupt Timing Diagram

MOTOROLA *Semiconductor Products Inc.*
Colvilles Road, Kelvin Estate - East Kilbride/Glasgow - SCOTLAND

Printed in Switzerland